Solving K-Set Partition Problem Using Genetic Algorithm

Supervised by

Dr. Mozammel Huq Azad Khan

Submitted by

Tarjia Alam Nisha
2010-3-60-003

Md. Shohan Ahmed
2010-3-60-008

East West University
Department of Computer Science and Engineering

Declaration

We hereby declare that, this project was done under CSE497 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose except for publication.

Tarjia Alam Nisha	Md. Shohan Ahmed
ID: 2010-3-60-003	ID: 2010-3-60-008

Letter of Acceptance

I hereby declare that this project is from the student's own work and best effort of mine, and all other source of information used have been acknowledged. This project has been submitted with my approval.

_____

Dr. Mozammel Huq Azad Khan **Supervisor**

Professor

Department of Computer Science and Engineering

East West University

_____

Dr. Shamim H Ripon **Chairperson**

Assistant Professor and Chairperson

Department of Computer Science and Engineering

East West University

Acknowledgement

Firstly our most heartfelt gratitude goes to our beloved parents for their endless support, continuous inspiration, great contribution and perfect guidance from the beginning to end.

We owe our thankfulness to my supervisor Dr. Mozammel Huq Azad Khan for his skilled, utmost direction, encouragement and care to prepare ourselves.

Our sincere gratefulness for the faculty of Computer Science and Engineering whose friendly attitude and enthusiastic support that has given us for four years.

We are very grateful for the motivation and stimulation from our good friends and seniors.

We also thank the researchers for their works that help us to learn and implement Genetic Algorithm for K-Set Partition Problem.

Abbreviation and Acronyms

SPP= Set Partition Problem

GA = Genetic Algorithm

LISP= List Processing

ACO= Ant Colony Optimization

Table of Contents

Declaration ii

Letter of Acceptance iii

Acknowledgement iv

Abbreviations and Acronyms v

List of Figures viii

List of Tables ix

List of Algorithm x

Abstract xi

Chapter 1:
Introduction 1-5

1.1 Introduction to proposed problem 1
1.2 Applications of SPP 1
1.3 Related Works 2
1.4 The set partitioning problem 3
1.5 Objective of the Research 4
1.6 Methodology of the Research 4

Chapter 2:
Introduction of Genetic Algorithm 6-8

2.1 Basics of Genetic Algorithm 6
2.2 Representation 7

Chapter 4:
Proposed Genetic Algorithm for k set Partition 9-16

4.1 Problem Representation 9

4.2 Proposed GA 10

4.3 Fitness Function 11

4.4 Deterministic Initialization of Population 12

4.5 Tournament Selection 13

4.6 Crossover 14

4.7 Mutation 15

4.8 Termination Condition 16

Chapter 5:
Experimental results and Analysis 17-20

5.1 Datasets used 17

5.2 Results and Discussion 17

Chapter 6:
Conclusions and Future Works 21

References 22-24

List of Figures

Figure 1:	Venn diagram for P, NP, NP-complete, and NP-hard set of problems	4
Figure 2:	GA Flow Chart	6
Figure 3:	Chromosome	10
Figure 4:	Crossover position Selected	14
Figure 5:	New offspring generated by crossover	14
Figure 6:	Average and best (minimum) fitness for William A. Greene's dataset	18

List of Tables

Table 1: Comparison of Obtained Results with reference no. [11] 18

Table 2: Experimental results of reference no. [10]'s datasets 19

Table 3: Experimental results of Created datasets 20

List of Algorithms

Algorithm 1: Basic Genetic Algorithm 7

Algorithm 2: Proposed GA procedure 11

Algorithm 3: Initialize population 13

Algorithm 4: Crossover 15

Algorithm 5: Mutation 15

Abstract

In this paper, we solved K-set partition problem with Genetic algorithm. K-set partition is a problem where we have to partition a given set of numbers into subsets such that their sums are as nearly equal as possible. In other hand, Genetic algorithm (GA) is a particular class of evolutionary algorithm that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.GA is implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. We present a GA in conjunction with a specialized heuristic improvement operator for solving K-set partition problem. The performance of our algorithm is evaluated on some set of real-world problems. Computational results show that the genetic algorithm-based heuristic capable of producing high quality solutions.

Chapter 1

Introduction

1.1. Introduction to proposed problem

The set partitioning problem (SPP) is a problem where we have to partition n numbers into k subsets, such that the sums of the subsets are as nearly equal as possible. The SPP is highly constrained optimization problem which offer greater challenges to solve effectively. Highly constrained problems may be thought of as those having a high proportion of equality constraints. These problems have traditionally been in the domain of specialized heuristics and integer linear programming techniques rather than of meta-heuristics [1]. This is because meta heuristics find it difficult to obtain and maintain feasible solutions to these problems. Specialized genetic operators were used to generate solutions [2]. In this paper GA is used as a medium to create solutions from which the feasibility restoration and improvement can be applied.

1.2. Application of SPP

The SPP is an extremely practical combinatorial optimization problem for which there exists a variety of applications (see Balas and Padberg [21] for an extensive overview). The most well known of these is the aircraft crew scheduling problem [22]. Other applications of SPP are truck scheduling (Balinski and Quant 1964), information retrieval (Day 1965), circuit design (Root 1964), capacity balancing (Steinman and Schwinn 1969), capital investment (Valenta 1969), facility location (Revelle, Marks and Liebman 1970), political districting (Garfinkel and Nemhauser 1970), and radio communication planning (Thuve 1981). The SPP has been the focus of study by many researchers because of its simple structure and numerous practical applications. We also feel interested to work with this problem because of its large amount of applications and simple structure.

1.3. Related work

Because of the importance of the SPP, a number of algorithms have been developed. These can be classified into two categories: exact algorithms which attempt to solve the SPP to optimality, and heuristic algorithms which try to find "good" solutions quickly. There are few heuristic solution algorithms for the SPP in the literature. This is because the SPP is a highly constrained problem and thus finding any feasible solution to a SPP is itself a difficult problem [3].In these days it is observed that noticeable applications of meta-heuristic approaches are used to solve SPP.

There have been relatively few attempts at using standard meta-heuristics to solve highly constrained problems such as the SPP. Abramson, Dang and Krishnamoorthy [1] are able to solve relatively small SPP instances to optimality using two variations of simulated annealing. Crawford and Castro [16]present an ACO algorithm that incorporates constraint programming lookahead functions that again is able to solve small problems. Due to the large amount of computational resources required, Levine [17] and Czech [18] have created parallel genetic algorithms and simulated annealing solvers respectively.

Marcus Randall's [2] binary ant colony optimization framework is applied to produce feasible solutions for SPP. This algorithms restore feasibility has been successfully applied to such problems. To increase its effectiveness, feasibility restoration, solution improvement algorithms and candidate set strategies are added. The overall results of this approach indicate that the ant colony optimization algorithm can efficiently solve small to medium sized problem.

According to Maniezzo and Milandri [15], the traditional ACO framework is ineffectual for solving the SPP. Hence they use a modified approach in which some elements of ACO are combined with tree search procedures. This becomes a form of branch and bound. At each step of their modified algorithm, the partial solution is expanded with up to k nodes (columns) per ant. The k nodes are produced with the tree search algorithm. Pheromone is

applied to the coupling of how well column i covers constraint/row j. The results show that such an approach is viable.

Chu and Beasley's [3] GA approach uses an augmented penalty method in order to produce feasible solutions. Rather than applying a simple penalty term to the fitness (objective) function; they separate this into two distinct functions, namely fitness and unfitness. The fitness is the original objective function while the unfitness function is the cumulative measure of the number of times each row is over or undercovered. Their GA consists of tailored genetic operators, parent selection techniques and a heuristic to improve the feasibility of child solutions (though not necessarily guarantee feasibility). The results showed that not only could feasible solutions be obtained, but many of these were optimal (even to large size problems).

1.4. The set partitioning problem (SPP)

The set partitioning problem (SPP) is to partition N numbers into K subsets, such that the sums of the subsets are as nearly equal as possible. The SPP is the following: given a set (or possibly a multiset) of N positive integers A = $\{a_1, a_2, \ldots, a_N\}$, find a partition $P\{1, \ldots ,N\}$ that minimizes the discrepancy

$$E(P) = \left| \sum_{i \in P} a_i - \sum_{i \notin P} a_i \right|$$

Partitions such that E = 0 or E = 1 are called perfect partitions. A decision problem related to the SPP is that of determining if there is a perfect partition [5].
The SPP is NP-hard problem. The relationship between P and NP problems is shown in Figure 1. The SPP is an NP-hard problem [6], which means that unless P = NP, there is no polynomial time algorithm for solving SPP. Since P = NP is widely believed to be false, it is unlikely that we can solve the problem in polynomial time using exact or approximation algorithms. Therefore, to solve SPP in polynomial time, various heuristic algorithms are used - such as greedy heuristics, simulated annealing, neural network and tabu search [6][7].

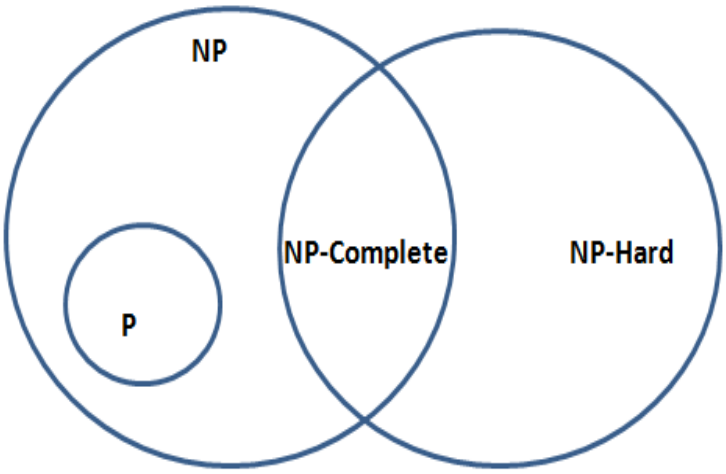


Figure 1: Venn diagram for P, NP, NP-complete, and NP-hard set of problems.

An important characteristic of this problem is that its computational complexity depends on the type of input numbers a_1, a_2 , …, a_N. Ifn is the number of elements in the input set and N is the sum of elements in the input set then this algorithm runs in time O(Nn) [4].

1.5. Objectives of the Research

Assume that we have a set of N numbers and we need to separate them into k subset where the differences between summations of subsets are equal or close to equal. Our target is to find the optimal solution to minimize the differences between subset summations. The problem is known to be NP-hard and we have developed a meta-heuristic algorithm based on Genetic algorithm (GA) to solve the problem. In this paper, we use GA combined with local optimizer which will take less time and better result than other approaches.

1.6. Methodology of the Research

In this paper, we propose a genetic algorithm (GA) that uses heap tree of the chromosomes to make initial population, classical crossover as main variation operation and probabilistic optimization technique of adding or removing bit to get better fitness result at the solution population. Therefore population of the solution is updated mainly using crossover operation

and heuristic improvement technique is applied on the offspring with low probability to locally improve the solution quality. It produces very good results compared to other well-known heuristic algorithms. In most of the cases it finds the nearest optimal solution. For a few cases it generates the exact solution.

Chapter 2

Introduction of Genetic algorithms

2.1. Basic Structure of Genetic Algorithm

Genetic Algorithm (GA) is developed based on the evolutionary process of biological organisms in nature. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals which are less fit will be eliminated. This means that the genes from the highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment [3]. Flow chart of GA is given in figure 2.

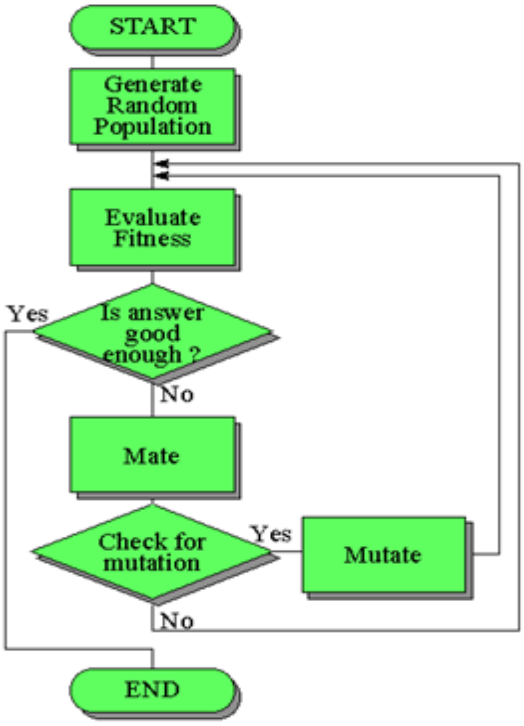


Figure 2: GA Flow Chart

A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimization terms, each individual in the population encoded into a string or chromosome which represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are given opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., children), which share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (generational approach) or replace less fit individuals (steady-state approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found [3]. The basic steps of a simple GA are shown below.

```
Generate an initial population;
Evaluate fitness of individuals in the population;
Repeat
        Select parents from the population;
        Recombine (mate) parents to produce children;
        Mutate children;
        Evaluate fitness of the children;
        Replace some or all of the population by the children;
Until a satisfactory solution has been found;
```

Algorithm 1: Basic Genetic Algorithm

2.2. Representation

Representing a chromosomes encoding varies depending on the problem nature. Traditionally, chromosomes are represented in binary as a string of 0s and 1s; however a number of different encoding can be used to represent a solution (chromosome). Besides binary encoding which is a string of 0s and 1s there is permutation encoding. In a permutation encoding, every chromosome is a string of numbers, which may represent a number in a sequence. A direct value encoding can be used in problems, where some

complicated value such as real numbers are used. In value encoding, chromosomes are represented using some value and this value can be anything connected to the problem, from numbers, real number, or chars to some complicated objects. Tree encoding can also be used for genetic algorithm, where each chromosome is a tree of some objects such as a function or commands in programming language. Tree encoding is good for evolving programs. Programming language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily [14].

Chapter 4

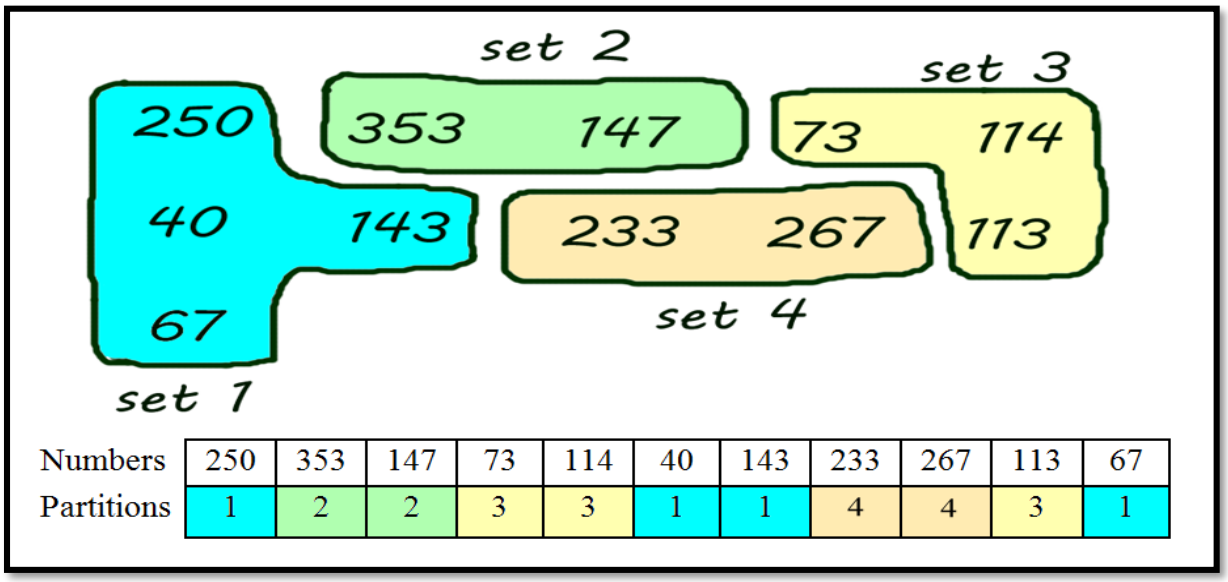
Proposed Genetic Algorithm for k set Partition

We propose hybrid genetic algorithm using general GA techniques with heap tree properties. The one of the main features of our GA is to use heap tree to generate initial population. To evaluate initial population we generate new generation using our fitness function and GA methods.

4.1. Problem Representation

Assume a problem that has to divide a set of numbers into a particular number of subset where additions of every subset numbers are almost equal to other subsets.
Suppose we have a set of 11 numbers.
Set, S= {250, 353, 147, 73, 114, 40, 143, 233, 267, 113, 67}
Now we need to divide this set into 4 subsets where additions of every subset numbers are almost equal.
$If\ S = \{S_1, S_{2,} S_{3,} S_4\}$
Our target is, $\sum S_1 \cong \sum S_2 \cong \sum S_3 \cong \sum S_4$
So its result will be:
$S_1 = \{250, 40, 143, 67\}$
$S_2 = \{353, 147\}$
$S_3 = \{73, 114, 113\}$
$S_4 = \{233, 267\}$

To solve this problem we randomly subset the numbers to generate some trivial solutions. From that trivial solution we try to get an optimal solution by evaluating these solutions by GA techniques. To do this we use permutation encoding technique [9], where every chromosome is a string of integer values. Every value indicates the subset number of that positioned number.



Numbers	250	353	147	73	114	40	143	233	267	113	67
Partitions	1	2	2	3	3	1	1	4	4	3	1

Figure 3: Chromosome

By permutation encoding technique every chromosome represents a solution of the problem. In Figure 3 we give a simple example of permutation encoding technique where partitions array represents the string of integer value that I have discussed above and partitions array is always parallel to number array. The value of partitions array will be at most the number of partitions we needed.

4.2. Proposed GA

We used heap tree search method as a local search tool. By using heap tree we generate initial population. Where each column of every chromosome represent a subset allocated numbers. From this initial population we generate new generation depending on our fitness function. In every generation, we replace the weakest chromosome by the fittest offspring by doing tournament selection, crossover, mutations and others. The structure of our GA's is shown in Algorithm 2.

Begin

Create an initialize population using heap tree;

While terminating condition is not true

begin

Select parent by tournament selection;

Generate two offspring by Crossover;

Mutate the two offspring;

Evaluate fitness of the offsprings;

Replace two weakest population by this offspring if they better;

Updater Terminating condition;

endfor

End

Algorithm 2: Proposed GA procedure

4.3. Fitness Function

The most important concept of genetic programming is the fitness function. The fitness function determines how well a program is able to solve the problem [19]. Fitness function evaluates how good a potential solution is relative to other potential solutions. The fitness function is responsible for performing this evaluation and returning a positive number or fitness value that reflects how optimal the solution is. In our problem low fitness value indicate better solution. Where the Chromosome fitness is the total difference between its every partition's allocated numbers sum and expected fitness. Its mathematical representation is given below.

Expected Fitness, $\bar{X} = \frac{\sum Total\ Numbers}{Number\ of\ Partitions}$

Fitness of ithChromosome, $\bar{F}_i = \sum_{i:=0}^{Totalpartition} \left| \sum Numbers\ in\ i^{th} partition - \bar{X} \right|$

Average Fitness of population, $\overline{A_{vg}} = \frac{\sum_{i:=0}^{Populationsize} |\bar{F}_i - \bar{X}|}{Populationsize}$

4.4. Deterministic Initialization of Population

The first step in the functioning of a GA is the generation of an initial population. Each member of this population encodes a possible solution to a problem. After creating the initial population, each individual evaluated and assigned a fitness value according to the fitness function. Each column of a Chromosome represents allocated subset of that positioned number.
To find a good initial population we use min heap tree in 30% of times and other 70% time we create population randomly. Heaps are a specialized tree-based data structure that used in applications concerned with priority queues and ordering. Heap can implement by two properties.

Min heap property: if X is a child of Y, then key(Y) ≤ key(X)
Max heap property: if X is a child of Y, then key(Y) ≥ key(X)

We work using min heaps because always we need to find out the smallest partition of the chromosome where summation of the allocated numbers is very low. In min heap tree smallest key is always at the root note because of the property of the element [20].
The time complexity of heap is given below, where N is total Number of node is tree. In our GA the value of N will be at most “The number of required partitions”.

Find min: $\Theta(1)$
Delete min: $\Theta(\log N)$
Insert Node: $\Theta(\log N)$

The way of our GA population initiating is shown in Algorithm 3.

For i:= 0 to Population Size

Initialize subsetSum[];

Deterministic:=randomly taken value from 1 to 100

If deterministic less than 30 then

Create min heap tree of partitions sum where every child represent a subset

Forj:=0 to Total Number

Begin:

k:=randomly taken value from 1 to 3

pop top k^{th} child from tree then allocate j^{th} number into this child and then insert this child into tree.

Update subsetSum[]

End for

End If

Else

forj:=0 to Total Number

Begin:

Randomly allocate j^{th} number into any subset.

Update subsetSum[]

End for

End else

End for

Algorithm 3: Initialize population

4.5. Tournament Selection

Tournament selection [13] is a variant of rank-based selection methods. Its principle consists in randomly selecting a set of k individuals. These individuals are then ranked according to their relative fitness and the fittest individual is selected for reproduction. Here we use Binary Tournament selection. First we randomly select two individual chromosomes from population then we chose best one as parents one and then again select two individual chromosomes from population and take best one as parent two.

4.6. Crossover

Crossover is the major instrument of variation and innovation in GAs. The simplest form of crossover is single-point crossover. It chooses a single crossover position at random and the parts of two parents after the crossover position are exchanged to form two offspring [8].

In figure 3 crossover positions selected of parent 1 & parent 2 at random.

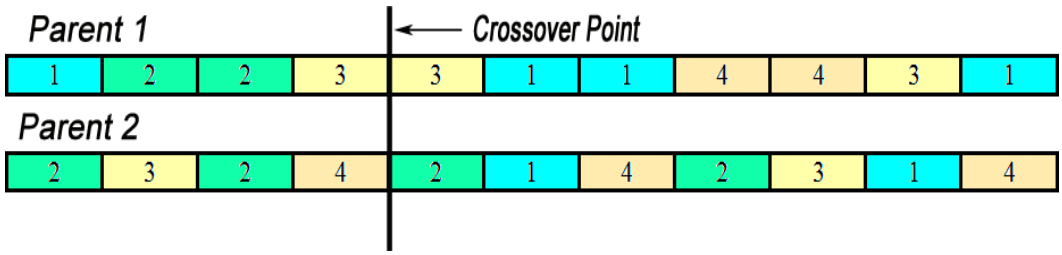


Figure 4: Crossover position Selected

In figure 4 gene of both side of crossover point are interchanges between the two parents to create two new offspring.

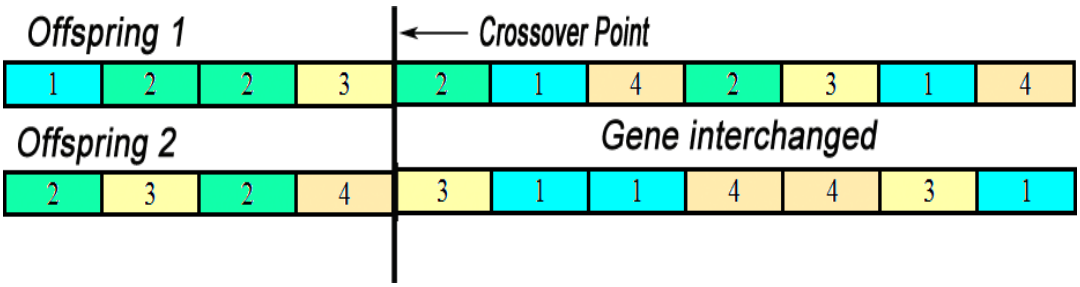


Figure 5: New offspring generated by crossover

We take crossover probability as 80% that means 80% time offspring made by crossover and other 20% time offspring made by exact copy of parents. Our crossover structure is shown in Algorithm 4.

Offspring1=paprent1
Offspring2=parent2
probability=randomly taken value from 1 to 100
if probability less than or equal 80 then
    k=randomly taken value from 1 to Total Number
    swap 1 to k^{th} value from offspring1 to offspring2
End if

Algorithm 4: Crossover

4.7. Mutation

A mutation operator randomly changes each character in a selected chromosome into another random character with probability Pm. The primary effect of mutation is to introduce new alleles, the values of genes, into the evolution process to avoid permanent fixed kinds of alleles in a population [7].In our generated population there is a high probability to create local optima because of using heap tree. So, to reduce this local optima we needed to give high mutation probability and that is 2%. Mutation structure of our GA's is shown in Algorithm 5.

For i=1 to total character of offspring
Begin:
    p=randomly taken value from 1 to 100
    Ifp is less than or equal mutation Probability
        k=randomly taken value from 1 to number of Partition
        Assign i^{th}character of this offspring is k
    End if
End

Algorithm 5: Mutation

4.8. Termination Condition

Termination is the criterion by which the genetic algorithm decides whether to continue searching or stop the search. Each of the enabled termination criterion is checked after each generation to see if it is time to stop. To make our solution batter we consider average fitness ($\overline{A_{vg}}$) of whole population. When average fitness remains unchanged for certain number of new generation then we decide to stop generating new population. Normally we waited 2N time where N is total numbers.

Chapter 5

Experimental results and Analysis

We have implemented our algorithm in C++ programming language utilizing GNU GCC Compiler and its random number generator is used for generating random numbers. We tasted 10 datasets and testes where run on a Laptop PC having flowing configuration:

CPU: Intel Core i3 2.13GHz
Memory: 6 GB DDR3 1333MHz
Operating System: Windows 7 64-bit

5.1. Datasets used

Datasets used to test our Genetic Algorithm for SPP are taken from Florida State University's partition_problem [10] and William A. Greene's Genetic Algorithms for Partitioning Sets [11]. We also create some dataset like spp79num10part, spp116num15part, spp151num20part and spp189num25part [12] to test our GA for SPP. The top value of the dataset means total numbers, second value means number of partitions and others are the numbers which we have to divide into k-partitions.

5.2. Results and Discussion

Result of our algorithm is compared with William A. Greene's Genetic Algorithms for Partitioning Sets [11] in Table 1. The table has four columns where first column shows in which section our algorithm is better; other three columns show the comparable results. For this dataset we set, 4*N to terminate the program where N is total number. The average and best fitness of for William A. Greene's dataset is shown in figure 5.

Table 1: Comparison of Obtained Results with reference no. [11].

Difference In	Falkenauer's Greedy GGA	Eager Breeder	Our GA Algorithm for SPP
Population size	50	250	50
Max number generations	3500	40	499
Trials finding optimal partition	30	29	0
Best fitness	9,608	3,242	3013
Implementation Stage	Theoretical	Theoretical	Implemented

Figure 6 shows the average fitness and minimum fitness over successive generation for William A. Greene's dataset which indicated the dynamicity of our algorithm.

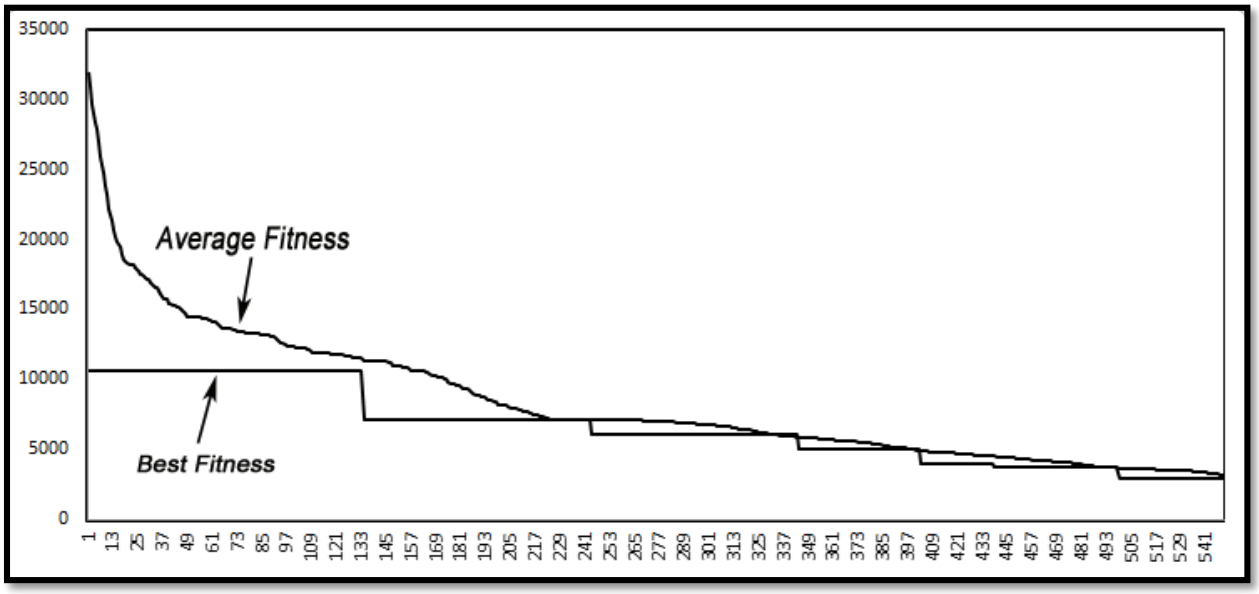


Figure 6: Average and best (minimum) fitness for William A. Greene's dataset

Our GA results of Florida State University's partition_problem [10] shown in table 2. The table has four columns where first column shows name of the datasets, second column shows total number of the datasets, third column shows number of partitions required, fourth column shows population size for each dataset, fifth column shows the number of generation need to get the best fitness and last column shows the fitness of our result.

Table 2: Experimental results of reference no. [10]'s datasets.

Dataset Name	Total Number	Partitions	Population Size	Generation	Fitness
p01_w	10	2	5	1	0
p02_w	10	2	10	51	24
P03_w	9	2	14	34	8
P04_w	5	2	10	10	4
P05_w	9	2	10	1	0

We are failed to find any satisfied large dataset for which we have created four large dataset to test our algorithm. In this dataset the range of number is large and number of partitions is also higher than other.

To create this dataset we use GNU GCC random generator. In each data first we give a partition number (P_n) that decide how many partitions will be in this data. Then P_n set of numbers generated where every set of numbers summation is exactly one thousand. To generate each set of P_n, randomly we take a number (N_p) that indicate how many numbers is in set P_n. Then randomly N_p of numbers are taken. If $\sum N_p$ is one thousand then we go for next partitions, otherwise we try again until we found $\sum N_p$ is one thousand. After finding all P_n set of numbers, we make a set(S) from P_n set of numbers.

Set, S= $\{P_1, P_2, P_3 P_4 \ldots \ldots . P_n\}$

Then finally we do ten thousand random swaps in set S to make it more promiscuous.

Test result of our created dataset shown in Table 3. The table has four columns where first column shows name of the datasets, second column shows total number of the datasets, third column shows number of partitions required, fourth column shows population size for each dataset, fifth column shows the number of generation need to get the best fitness and last column shows the fitness of our result.

Table 3: Experimental results of Created datasets

Dataset Name	Total Number	Partitions	Population Size	Generation	Fitness
spp79num10part	79	10	50	2912	8
spp116num15part	116	15	80	9202	40
spp151num20part	151	20	80	16911	78
spp189num25part	189	25	190	81984	108

Chapter 6

Conclusions and Future Works

In this paper, we propose a genetic algorithm (GA) for k-set partition problem (SPP). Unlike the previous works, we use heap tree of the chromosomes to make initial population in GA for the first time for SPP. The main variation operator of our GA is the classical crossover operator of the genetic algorithm (GA). That means the population of the solutions is updated mainly using crossover operator. We select two parents randomly and apply the crossover operator with a high probability. Due to the nature of the encoding, the generated offspring's may become invalid and in that case the offspring's are corrected to valid solution. Then a deterministic improvement technique is applied on the corrected offspring's with low probability to locally improve the solution quality. If the generated offspring's is better than the worst solution is replaced by offspring. The combination of the genetic operation and the deterministic improvement makes the algorithm hybrid.

We test 10 datasets and compare with William a. Greene's genetic algorithms for partitioning sets [11] and our solutions are the best solution till now. We obtained better solutions over recent works for the datasets of Florida State University's partition_problem [10]. Our algorithm gives the best solution for very spp79num10part, spp116num15part, spp151num20part and spp189num25part [12]our meta-heuristic solution takes very small time compared to others deterministic algorithms to reach the optimal solutions. Overall, it provides very good solutions, comparable to the best algorithms that have been proposed so far.

In the future work, we will try to use better data structure and improve the local optimizer. We have also plans to compare the results of our proposed approach with the results proposed by other evolutionary algorithm.

REFERENCES:

[1] D. Abramson, H. Dang, and M. Krishnamoorthy, " A comparison of methods for solving 0-1 integer programs using a general purpose simulated annealing algorithm," Annals of Operations Research, 63:129–150, 1996.

[2] Marcus Randall and Andrew Lewis, " Modifications and additions to ant colony optimization to solve the set partitioning problem," IEEE Computer Society, 978-0-7695-4295-9/10, 2010.

[3] P.C Chu and J. E. Beasley, "Constraint Handling in Genetic Algorithms: The Set Partitioning Problem," Journal of Heuristics, 11:323-357(1998).

[4] Stephan Mertens, " The easiest hard problem: Number partitioning," In A.G. Percus, G. Istrate, and C. Moore, editors, Computational Complexity and Statistical Physics, pages 125-139, New York, 2006, Oxford University Press.

[5] Jo~ao Pedro Pedroso and Mikio Kubo, " Heuristics and exact methods for number partitioning," Technical Report Series: DCC-2008-3.

[6] P.M.Pardalos and J.Xue, "The Maximum Clique Problem," Journal of Global Optimization, 4, 1994,pp.301-328.H. Poor, An Introduction to Signal Detection and Estimation, *Springer-Verlag*, New York, ch. 4, 1985.

[7] Bo Huang, "Finding Maximum Clique with a Genetic Algorithm", The Pennsylvania State University,Master's Paper, 2002.

[8] M. Mitchell, "An Introduction to Genetic Algorithms," The MIT Press, Cambridge, MA, 1999.

[9] S.N. Sivanandam, S. N. Deepa , "Terminologies and Operators of GA," in Introduction to Genetic Algorithms, 1st Edition, New York : Springer, 2007, pp-41-46

[10] Florida State University, in Tallahassee, Florida, (09 May 2012). partition_problem [Online]. Available FTP: http://people.sc.fsu.edu/~jburkardt/datasets/partition_problem/partition_problem.html (Accessed: 20 July 2014).

[11]William A. Greene, "Genetic Algorithms For Partitioning Sets," World Scientific Publishing Company, Singapore, Vol 10, Nos. 1&2. pp. 225-241.

[12] Md. Shohan Ahmed (29 November 2014). K-set partitions Dataset [Online] Available FTP: https://www.dropbox.com/sh/694mqixkfrezr39/AAA3ule78gZPqlA7ugVcBAC4a?dl=0

[13]K. Hingee. K, Hutter. M,"Equivalence of probabilistic tournament and polynomial ranking selection," In Evolutionary Computation, 2008, CEC 2008.(IEEE World Congress on Computational Intelligence), IEEE Congress on 2008; 564-571.

[14]Lenin Hasda and FariaSabnam, "Solving Degree Constrained Minimum Spanning Tree Problem Using Genetic Algorithm," East West University, Undergrad paper, 2014.

[15] V. Maniezzo and M. Milandri, "An ant-based framework for very strongly constrained problems," In M. Dorigo, G. Di Caro, andM. Sampels, editors, Third International Workshop on Ant Algorithms, ANTS 2002, volume 2463 of Lecture Notes in Computer Science, pages 222–227, Brussels, Belgium, 2002, Springer-Verlag.

[16]B. Crawford and C. Castro, "ACO with lookahead procedures for solving set partitioning and covering problems," Workshop proceedings of "Combination of Metaheuristic and Local Search with Constraint Programming Techniques", 2005.

[17] D. Levine, "A parallel genetic algorithm for the set partitioning problem," In I. Osman and J. Kelly, editors, Meta-Heuristics: Theory and Application, pages 23–35., Kluwer Academic Publishers, Norwell, MA, 1996.

[18] Z. Czech, "A parallel genetic algorithm for the set partitioning problem,"
In 8^{th} Euromicro Workshop on Parallel and Distributed Processing, pages 343–350, 2000.

[19] Jaime J. Fernandey .The GP Tutorial [Online].
Available FTP: http://www.geneticprogramming.com/Tutorial/#anchor146647
(Accessed: 2 December 2014).

[20] Soheil Pourhashemi, "Skew Heap, Leftist Heap and Alternatives," York University, CSE PROJECT, 2007.

[21] E. Balas and M. Padberg, "Set Partitioning - A Survey. In N. Christophides," A. Mingozzi, P. Toth, and C. Sandi, editors, Combinatorial Optimization, pages 151–210, John Wiley, 1979.

[22] K. Hoffman and M. Padberg, "Solving airline crew-scheduling problems by branch-and-cut," Management Science, 39:657–682, 1993.